IIC3272 — Criptomonedas y contratos inteligentes

Tarea 1: Arboles de Merkle

Entrega: Abril 30

1. Cosas administrativas

Cada tarea en este curso equivale a 5 % de la nota final. En total vamos a tener 5 tareas, pero la peor tarea que solucionan no cuenta para la nota final. Quiere decir que pueden botar una tarea.

Para esta tarea van a tener que crear un programa en Python para implementar ciertas funcionalidades en arboles de Merkle.

Su programa va a extender nuestra librería BitcoinMerkle.py, disponible en canvas junto con este enunciado.

El la librería BitcoinMerkle.py hay ciertos métodos que no son implementados, y su trabajo consta en implementar los métodos, y enviarnos su programa con la solución. Nosotros vamos a diseñar una secuencia de datos de prueba para probar si su implementación funciona bien.

El programa con su solución hay que enviar a los siguiente dos correos (**si, a los dos al mismo tiempo por favor**):

- dvrgoc@ing.puc.cl
- rrrodriguez@uc.cl

Uso de materiales (o soluciones) encontradas en Internet está permitido. Solo se les pide citar la fuente que utilizaron. No se aplica ninguna penalidad para su uso.

2. La tarea

En esta tarea tienen que implementar tres nuevos métodos en la librería BitcoinMerkle.py, y una nueva clase. Los métodos y l clase son lo siguiente:

1. `def generate_proof(self, hashesOfInterest):`

   Este método es un método de la clase MerkleTree, y el método devuelve la prueba que se necesita para construir un raíz de Merkle en el método `populate_tree(self,`

`flag_bits, hashes)`. Esto quiere decir que el método devuelve la secuencia de bits `flag_bits`, y la secuencia de hashes `hashes`, necesaria para verificar si los hashes en la lista `hashesOfInterest` realmente pertenecen a este árbol de Merkle.

Pueden asumir que hashes en `hashesOfInterest` están hojas del árbol de Merkle.

El objeto que devuelve el método debería ser de la clase:

```
class MerkleProof:
    def __init__(self, hashesOfInterest, nrNodes=None,\\
                       flags=None, hashes=None):
        self.hashesOfInterest = hashesOfInterest
        self.nrNodes = nrNodes
        self.flags = flags
        self.hashes = hashes
```

Esta clase no tiene ningún otro método, y solo sirve para guardar una prueba en un árbol de Merkle. Como pueden ver, esta prueba incluye toda la información necesaria para verificar si los hashes en `hashesOfInteres` pertenecen al árbol de Merkle con cierta raíz `Mroot` (que no está en la clase, porque se supone que la persona buscando la demuestra ya conoce al raíz). Noten que para esto necesitamos también anunciar el número de nodos `nrNodes` en nuestro árbol de Merkle.

Nota: Es posible que van a tener que ampliar la clase `MerkleTree` tal que incluya todos los niveles del árbol construido al recibir hashes de hojas. Para esto se puede modificar el código de la clase `PartialMerkleTree`, o pensar en una implementación nueva.

2.
```
class SortedTree:

        .
        .
        .

    def proof_of_non_inclusion(self,hash):
```

La clase `SortedTree` vamos usar para poder generar pruebas de non-inclusion de ciertos hashes en nuestro árbol de Merkle. Para esto, La clase debería construir un árbol de Merkle ordenado. Los hashes se deberían ordenar según su valor en el formato hexadecimal (ojo que construcción del árbol de Merkle igual requiere estos valores en bytes). Es su decisión como implementar distintos métodos de la clase que van a ocupar para construir el árbol de Merkle.

El método `proof_of_non_inclusion(self,hash)` recibe como su entrada un hash y debería devolver la prueba que `hash` recibido como el input no pertenece al árbol de Merkle (i.e. al objeto `self`). La prueba debería ser un objeto de la clase `MerkleProof`.

Si lo ven necesario, pueden definir una nueva clase para esto, pero `MerkleProof` ya sirve.

3. **def** verify_non_inclusion(**hash**, merkleRoot, proof):

   Este método recibe como su input un hash `hash`, y una prueba `proof` de no-inclusión de `hash` en el árbol de Merkle con la raíz `merkleRoot`, que se también recibe como el input. El método debería verificar si la prueba sirve o no (i.e. devuelve `True` si `hash` **no pertenece** al árbol de Merkle con la raíz `mekrleRoot`, y `False` si pertenece, o si la prueba está mala.).

Puntos. La asignación de los punto es la siguiente:

- `generate_proof` – 5 puntos
- `SortedTree` y `proof_of_non_inclusion` – 1 punto
- `verify_non_inclusion` – 1 punto.